# Numerical Solution to Differential Equations using Matlab: Part 3: a finite element implementation for a 2D elliptic problem

Niall Madden

28 March 2012

## Abstract

Here is a short description of Matlab implementation of a finite element method for a two-dimensional problem. It presents my own approach (or, rather, a simplification of that). However, parts of the implementation and description are based on the excellent presentation given in [1]. If you are interested in developing a more expensive implementation, I suggest you consult that book.

## 1 Outline

Following on from last week's lab, today we want to develop some Matlab code to implement a simple finite element methods for the two-dimensional Poisson problem:

$$-(u_{xx} + u_{yy}) = f(x,y) \quad x \in \Omega := (0,1)^2, \qquad (1)$$

and with Dirichlet boundary conditions. Here are the limits of this code, and how it could be extended:

1. It is for the problem $-\Delta u = f$. It can be extended to the more general cases such as $-\Delta u + \mathbf{p}\cdot\nabla u + r(x,y)u = f$, or $-\nabla\cdot(\kappa\nabla u) = f$, where $\kappa$ is variable.
2. I use linear basis functions on triangles. It would take a little more work to extend to more general elements.
3. We could implement *mixed* boundary conditions. That is, we could have Dirichlet boundary conditions on one part of the domain, and Neumann (for example) on another.
4. The mesh is uniform. However, that is just for generating nice-looking results. It works for an arbitrary set of mesh points, which are then triangulated using Matlab's `delaunay` function.
5. The method is implemented as a script file. It would be better to implement this as a function.
6. It is rather slow. I haven't used any fancy sparse constructs to build the system matrix. That would not be hard to do, especially for a simple problem like this one, but it would make the code much harder to understand.
7. The simplest possible, one-point quadrature rule is used. On the triangle, $T_k$, we estimate the integral
$$\int_{T_k} g(x,y)dA \approx \bar{g}\text{Area}(T_k).$$
where $\bar{g}$ is the value of $g$ at the centroid of $T_k$. This is fine for constructing the element matrix, but is a little crude for the right-hand side.
8. The maximum pointwise error is reported. It should estimate the error in the energy norm.

## 2 The code

You can download the code from
`http://www.maths.nuigalway.ie/ niall/MathsOfFEM`

### 2.1 Generic code comments

As ever, our program begins with a short description of what the program does and why it was written.

```
%% An implementation of a Finite Element Method
% Author: Niall Madden. Date: 28/03/12
% for a two-dimensional PDE with linear basis
% elements on triangles. The problem is:
%  -(u_xx + u_yy) + r u=f(x,y) on Omega=(0,1)^2
% u given on boundary
```

### 2.2 Problem specific information

The test problem is simple variant on (1):

$$-\Delta u = 32(x - x^2 + y - y^2) \text{ on } \Omega, \quad u(\partial\Omega) = 0, \quad (2)$$

where $\Omega = (0,1)^2$ is the unit square. The exact solution is $u(x,y) = 16xy(1-x)(1-y)$, and is shown in Figure 1.

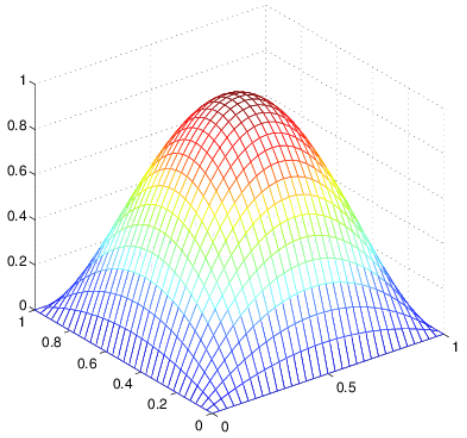

Figure 1: The solution to (2)

```
17  u=@(x,y)(16*x.*y.*(1-x).*(1-y));
18  uxx=@(x,y)-32*(y.*(1-y));
19  uyy=@(x,y)-32*(x.*(1-x));
20  f=@(x,y)(-uxx(x,y) - uyy(x,y) );
```

## 2.3 The mesh

We'll choose a fixed number $N$ of mesh intervals on each axis, and let `x` and `y` be vectors of $N+1$ mesh points on $[0,1]$. We then make a two-dimensional grid of $(N+1)^2$ mesh points from these by taking the Cartesian product. This is done by using the Matlab function `meshgrid`. It returns two $(N+1)\times(N+1)$ matrices, which are reshaped as vectors. This works by stacking the columns of the matrices to gives two (column) vectors `X` and `Y` with $(N+1)^2$ entries. Now vertex `i` will have coordinates $(X(i), Y(i))$. Here the mesh nodes are numbered left-to-right and top-to-bottom ("lexicographical ordering"). That is, node `1` is in the bottom left at $(0,0)$, node $N+1$ is in the bottom right at $(1,0)$, and note $(N+1)^2$ is in the top right at $(1,1)$. So, for example, `xk = X(T(k,:))` and `yk = Y(T(k,:))` assigns the vectors $x_k$ and $y_k$ the coordinates of the three vertices of triangle `k`.

```
30  N=2^3;
31  x = linspace(0,1,N+1); y=x;
32  [X, Y] = meshgrid(x,y);
33  X = reshape(X',[],1);
34  Y = reshape(Y',[],1);
```

The next step is the *triangulate* the data. This could be difficult, but there is a built in Matlab function called `delaunay` to help us. It returns a $(N+1)^2\times 3$ matrix. The $k^{\text{th}}$ row of this matrix will have the indices of the three vertices of triangle `k`. Such a mesh for the case $N=8$ is shown in Figure 2 below. There is a more sophisticated (and better) Matlab function called `DelaunayTri`, but I don't want to get into object oriented programming here.

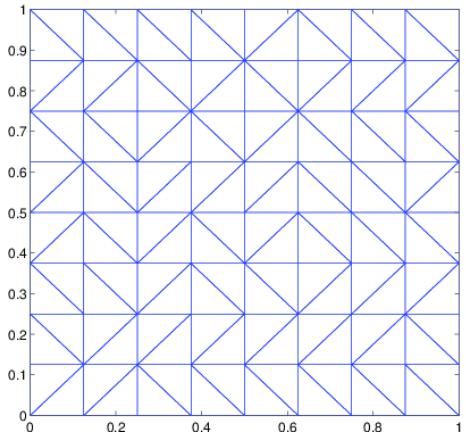

Figure 2: The triangulated mesh with $N=8$

```
37  T = delaunay(X,Y);
38  figure(1); triplot(T,X, Y );
```

## 2.4 The basis elements

Our basis is the set of piecewise linear functions on mesh:

$$\{\phi^{(1)}(x,y),\phi^{(2)}(x,y),\ldots,\phi^{\left((N+1)^2\right)}(x,y)\},$$

where here I am indexing from 1 since that is what Matlab does. Each of these can be defined as being the piecewise linear function such that

$$\phi^{(k)}(x_i,y_j)=\begin{cases}1 & x_i=X(k),y_j=X(k)\\ 0 & \text{otherwise.}\end{cases}$$

Unlike in the one-dimensional case, we don't tend to think of an arbitrary basis function in terms of its support (that is, the triangles where it is non-zero). This is because any number of triangulations are possible on a set of nodes. Since each basis function corresponds to a node, many triangulations are possible. Moreover, on a given (even uniform) arrangement of mesh nodes, different basis functions can have support on a different number of triangles. For example, looking at Figure 2 you'll see that the basis function associated with the mesh point $(2/4,1/8)$ has support on seven triangles, whereas the one associated with $(2/8,6/8)$ has support only on four.

Instead, it make much more sense the consider each triangle, and then determine the formula for the three basis functions that have support on it. The trick is

to find an easy way of determining the coefficients. We also need compute the area of each triangle, since this is useful for the numerical quadrature procedure.

We dealing first with the area of the triangle, I'll following the explanation and notation of [1, Section 4.6]. Suppose I have an arbitrary triangle $T_k$ with vertices $(x_1, y_1)$, $(x_2, y_2)$ and $(x_3, y_3)$. This can be mapped to the *reference triangle* $T_R$ on the vertices $(s_1, t_1) = (0, 0)$, $(s_2, s_2) = (1, 0)$ and $(s_3, t_3) = (0, 1)$. The mapping can be done by

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + J_k \begin{pmatrix} s \\ t \end{pmatrix} \text{ where } J_k = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}.$$

Then the area of triangle $k$ is $A_k = |J_k|/2$.

Now we will compute the coefficients of the three basis functions on the triangle $T_k$. The description here follows [1, Section 7.1]. I will denote by $\phi_1$, $\phi_2$ and $\phi_3$ the restriction of the basis functions that have support on $T_k$ to $T_k$. Each is associated with a vertex of $T_k$, in the sense that $\phi_1$ takes the value 1 at $(x_1, y_3)$, and is zero at $(x_2), y_2)$ and $(x_3, y_3)$. Then one could write

$$\phi_i(x, y) = a_i + b_i x + c_i y, \qquad \text{for } i = 1, 2, 3.$$

We have three equations and three unknowns:

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 2 & x_2 & y_2 \\ 3 & x_3 & y_3 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Lets write this as

$$M_k \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = e_1.$$

This is trivial to form and solve in Matlab. In particular, I form $M_k$ by `Mk = [ones(3,1), xk, yk]`.

Now the coefficients for $\phi_2$ and $\phi_3$ can be similarly determined.

$$M_k \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = e_2, \quad M_k \begin{pmatrix} a_3 \\ b_3 \\ c_3 \end{pmatrix} = e_3.$$

So writing

$$C_k = \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

we see than $C_k = M_k^{-1}$.

## 2.5 Building the linear system

Our linear system is

$$AU = F,$$

where $A_{i,j} = \int_\Omega \nabla\phi^{(i)} \cdot \nabla\phi^{(j)} dA$.

Rather than building $A$ one entry at a time, we iterate over each triangle and accumulate the integrals.

This is particularly easy in this case since the gradients are constant: $\nabla\phi_i = (b_i, c_i)^T$.

The full code is given as

```
41 A = sparse( (N+1)^2, (N+1)^2);
42 F = zeros( (N+1)^2, 1);
43 NT = length(T);
44 for k=1:NT
45   xk = X(T(k,:));
46   yk = Y(T(k,:));
47
48   Jk = [xk(2)-xk(1), xk(3)-xk(1);
49     yk(2)-yk(1), yk(3)-yk(1)];
50
51   Ak = abs(det(Jk))/2;
52   Mk = [ones(3,1), xk, yk];
53   Ck = inv(Mk);
54
55   E2 = zeros(3,3);
56   for i=1:3
57     for j=1:3
58       E2(i,j) = Ak*Ck(2:3, i)'*Ck(2:3, j);
59     end
60   end
61   A(T(k,:), T(k,:)) = A(T(k,:), T(k,:)) + E2;
62
63   Centroid_x = sum(xk)/3;
64   Centroid_y = sum(yk)/3;
65   F(T(k,:)) = F(T(k,:)) + ...
66       Ak*f(sum(xk)/3, sum(yk)/3)/3;
67 end
```

## 2.6 Solve and display

```
72 Boundary = find((X==0) | (X==1) | (Y==0) | (Y==1));
73 Interior = setdiff(1:(N+1)^2, Boundary);
74 U = zeros( (N+1)^2, 1);
75 U(Boundary) = u(X(Boundary) , Y(Boundary) );
76 F(Interior) = F(Interior) - ...
77     A(Interior, Boundary)*U(Boundary);
78 U(Interior) = A(Interior, Interior)\F(Interior);
79
80 figure(3);
81 subplot(1,3,1); trimesh(T,X,Y,u(X,Y));
82 subplot(1,3,2); trimesh(T,X,Y,U);
83 subplot(1,3,3); trimesh(T,X,Y,u(X,Y)-U);
```

# References

[1] Mark S. Gockenbach. *Understanding and implementing the finite element method.* SIAM, 2006.